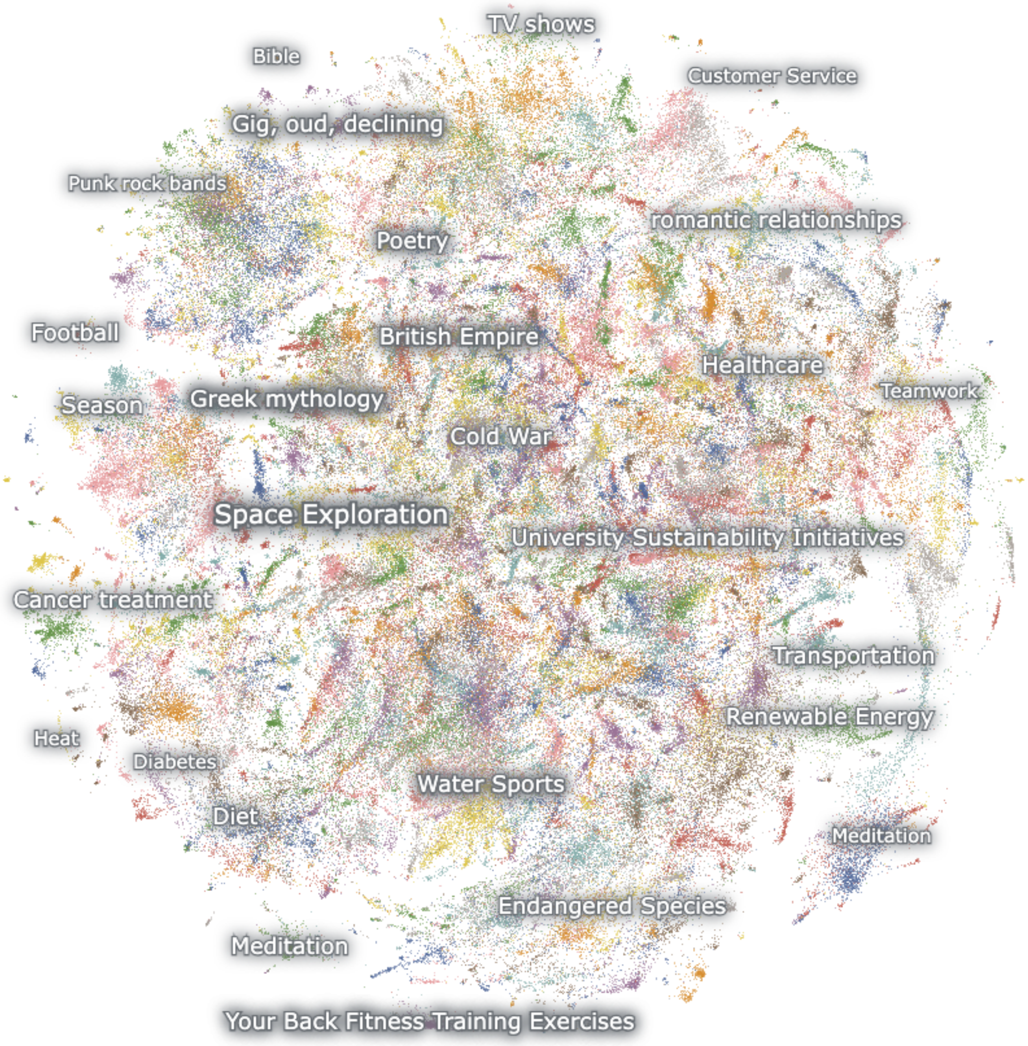
Deep dive into LLMs like ChatGPT by Andrej Karpathy (TL;DR)

February 8, 2025



Who is this deep dive for?

A few days ago, Andrej Karpathy released a video titled "Deep dive into LLMs like ChatGPT." It’s a goldmine of information, but it’s also 3 hours and 31 minutes long. I watched the whole thing and took a bunch of notes, so I figured why not put together a TL;DR version for anyone who wants the essential takeaways without the large time commitment.

If any of this sounds like you, this post (and the original video) is worth checking out:

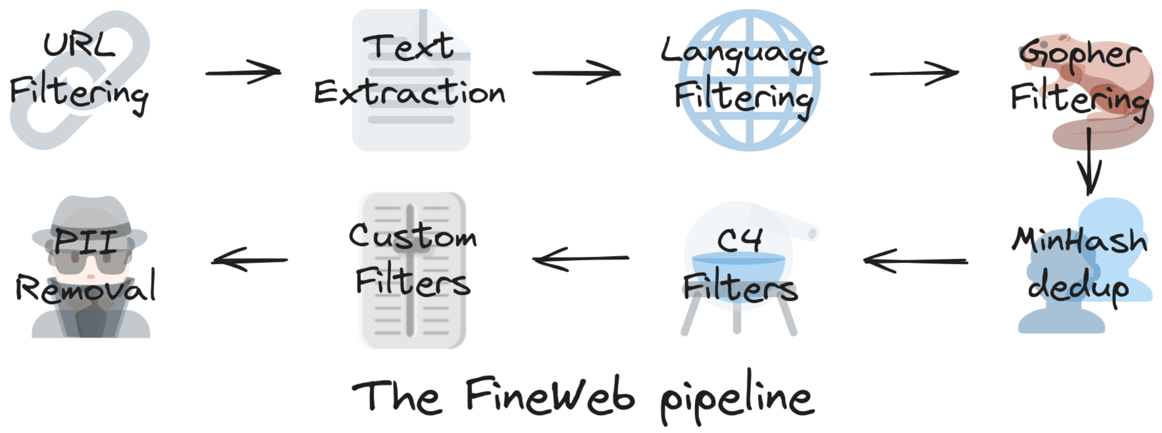
- **You want to understand how LLMs actually work** not just at the surface level.
- **You want to understand confusing fine-tuning terms** like `chat_template` and `ChatML` (especially if you're using Axolotl).
- **You want to get better at prompt engineering** by understanding why some prompts work better than others.
- **You're trying to reduce hallucinations** and want to know how to keep LLMs from making things up.
- **You want to understand why DeepSeek-R1 is such a big deal right now**.

I won't be covering *everything* in the video, so if you have time, definitely watch the whole thing. But if you don't, this post will give you the key takeaways.

Note: If you are looking for the excalidraw diagram that Andrej made for the video, you can download it here. He shared it through Google Drive and it invalidates the link after a certain time. That's why I have decided to host it on my CDN as well.

Pretraining Data

Internet



LLMs start by crawling the internet to build a massive text dataset. The problem? Raw data is noisy and full of duplicate content, low-quality text, and irrelevant information. Before training, it needs heavy filtering.

- If you're building an English only model, you'll need a heuristic to filter out non-English text (e.g., only keeping text with a high probability of being English).
- One example dataset is FineWeb, which contains over 1.2 billion web pages.

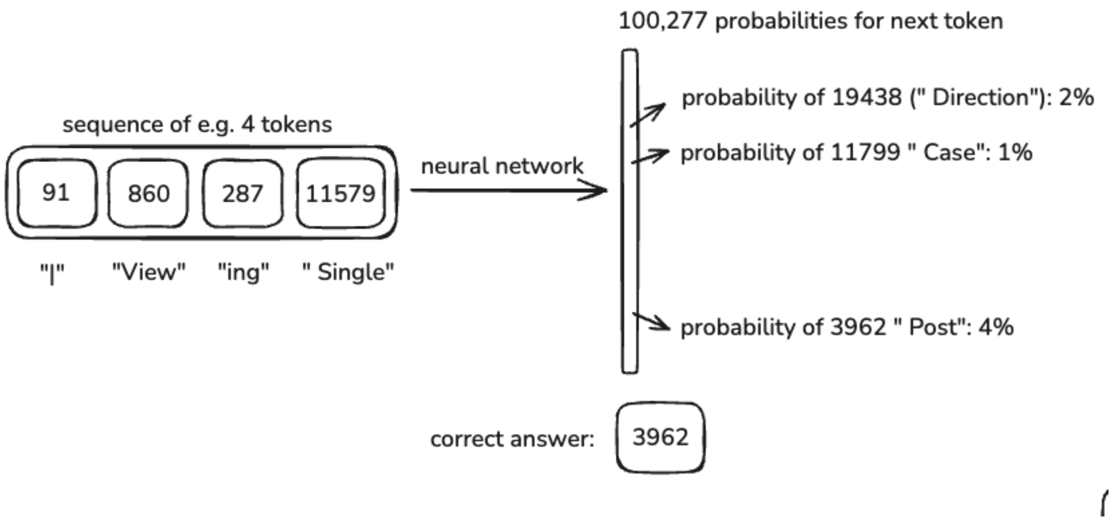
Once cleaned, the data still needs to be compressed into something usable. Instead of feeding raw text into the model, it gets converted into tokens: a structured, numerical representation.

Tokenization

Tokenization is how models break text into smaller pieces (tokens) before processing it. Instead of storing raw words, the model converts them into IDs that represent repeating patterns.

- A popular technique for this is Byte Pair Encoding (BPE).
- There's an optimal number of symbols (tokens) for compression. For example, GPT-4 uses **100,277** tokens. It is totally dependent on the discretion of the model creator.
- You can visualize how this works using tools like Tiktokenizer.

Neural Network I/O

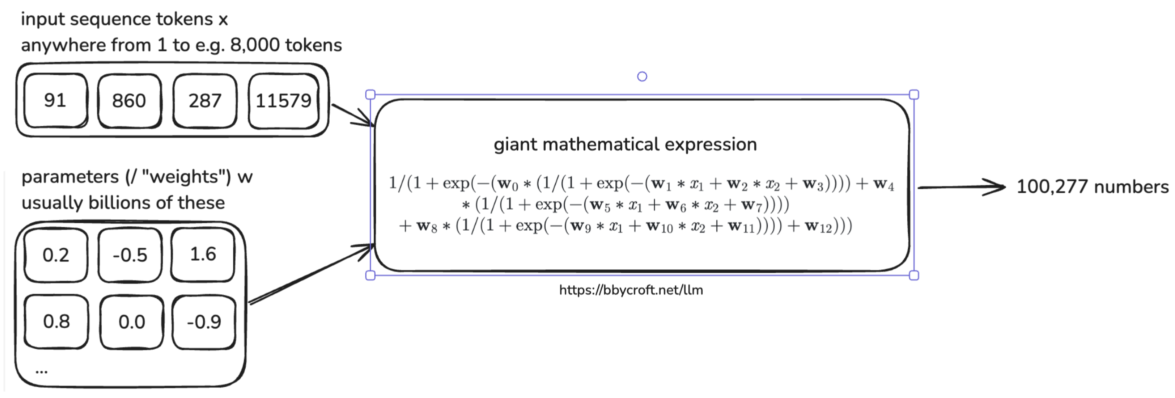

100,277 probabilities for next token
sequence of e.g. 4 tokens
91
860
287
11579
"|"
"View"
"ing"
" Single"
neural network
probability of 19438 (" Direction"): 2%
probability of 11799 " Case": 1%
probability of 3962 " Post": 4%
correct answer:
3962

Once the data is tokenized, it's fed into the neural network. Here's how that process works:

1. The model takes in a **context window**, a set number of tokens (e.g., 8,000 for some models, up to 128k for GPT-4).
2. It predicts the **next token** based on the patterns it has learned.
3. The weights in the model are adjusted using **backpropagation** to reduce errors.
4. Over time, the model learns to make better predictions.

A longer context window means the model can "remember" more from the input, but it also increases computational cost.

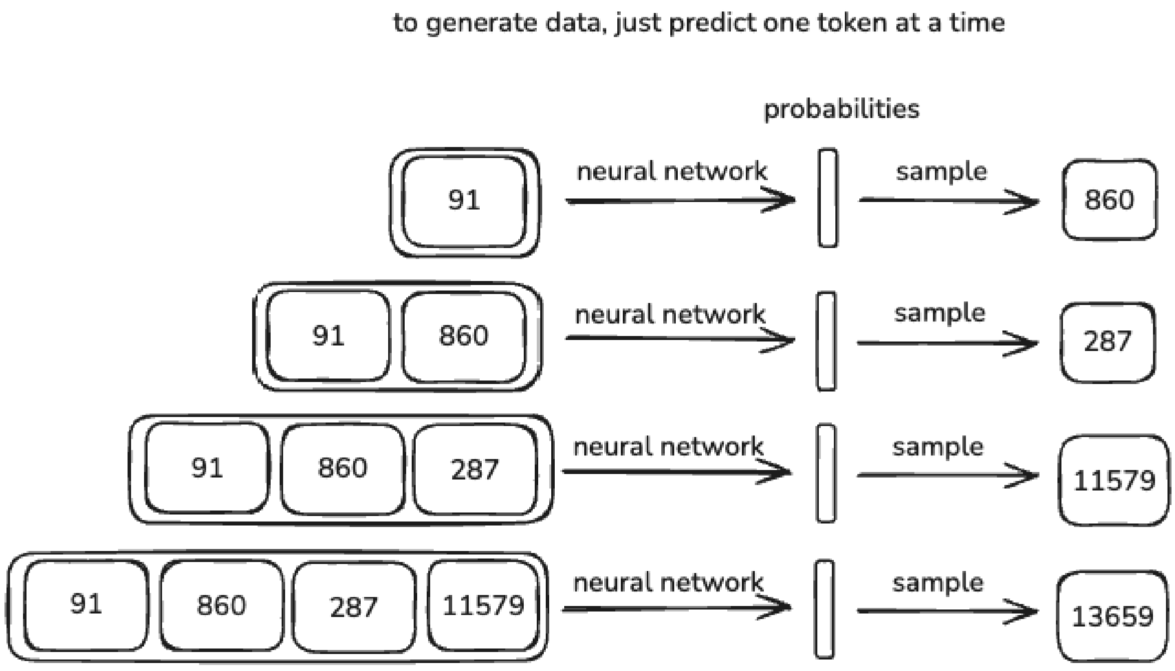
Neural Network Internals


input sequence tokens x
anywhere from 1 to e.g. 8,000 tokens
91
860
287
11579
parameters (/ "weights") w
usually billions of these
0.2
-0.5
1.6
0.8
0.0
-0.9
...
giant mathematical expression
$1/(1+\exp(-(\mathbf{w}_0 * (1/(1+\exp(-(\mathbf{w}_1 * x_1 + \mathbf{w}_2 * x_2 + \mathbf{w}_3)))) + \mathbf{w}_4$
$* (1/(1+\exp(-(\mathbf{w}_5 * x_1 + \mathbf{w}_6 * x_2 + \mathbf{w}_7))))$
$+ \mathbf{w}_8 * (1/(1+\exp(-(\mathbf{w}_9 * x_1 + \mathbf{w}_{10} * x_2 + \mathbf{w}_{11})))) + \mathbf{w}_{12})))$
https://bbycroft.net/llm
100,277 numbers

Inside the model, billions of parameters interact with the input tokens to generate a probability distribution for the next token.

- This process is defined by complex mathematical equations optimized for efficiency.
- Model architectures are designed to balance speed, accuracy, and parallelization.
- You can see a production-grade LLM architecture example here.

Inference



LLMs don’t generate deterministic outputs, they are **stochastic**. This means the output varies slightly every time you run the model.

- The model doesn’t just repeat what it was trained on, it generates responses based on probabilities.
- In some cases, the response will match something in the training data exactly, but most of the time, it will generate something new that follows similar patterns.

This randomness is why LLMs can be creative, but also why they sometimes **hallucinate** incorrect information.

GPT-2

GPT-2, released by OpenAI in 2019, was an early example of a transformer-based LLM.
Here's what it looked like:

- **1.6 billion parameters**
- **1024-token context length**
- **Trained on ~100 billion tokens**
- The original GPT-2 training cost was **$40,000**.

Since then, efficiency has improved dramatically. Andrej Karpathy managed to **reproduce GPT-2** using llm.c for just **$672**. With optimized pipelines, training costs could drop even further to **around $100**.

Why is it so much cheaper now?

- **Better pre-training data extraction techniques** → Cleaner datasets mean models learn faster.
- **Stronger hardware and optimized software** → Less computation needed for the same results.

Open Base Models

Disclaimer: The models under discussion **do not strictly follow** the Open Source Initiative (**OSI**) definitions of open-source AI (OSI AI Definition). Instead, we use the term **open base models** to describe models where the **weights are publicly accessible**, but **training data and full reproducibility** may not be provided.

Some companies train massive **language models (LMs)** and **release the base models for free**. A **base model** is essentially a **raw, pre-trained LM**—it still requires **fine-tuning** or **alignment** to be practically useful.

- Base models are trained on **unfiltered internet-scale data**, meaning they generate **raw** completions but **lack alignment with human intent**.
- OpenAI released **GPT-2**, an **open-weight and source-available** model, but **not fully open-source** under OSI's definition since its training data was not released.
- Meta released **Llama 3.1 (405B parameters)**, an **open-weight** but **not open-source** model.

To release a **base model**, two key components are required:

1. **Inference code** → Defines the steps the model follows to generate text. Typically written in Python.

2. **Model weights** → The billions of parameters that encode the model's knowledge.

How Base Models Work

- They generate **token-level** internet-style text.
- Every run produces **a slightly different output** (stochastic behavior).
- They can **regurgitate** parts of their training data.
- The parameters are like **a lossy zip file** of internet knowledge.
- You can already use them for applications like:
    - **Translation** → Using in-context examples.
    - **Basic assistants** → Prompting them in a structured way.

Want to experiment with one? Try the **Llama 3 (405B base model)** here.

At its core, a base model is **just an expensive autocomplete**. It still needs fine-tuning.

Pre-Training to Post-Training

So far, we've looked at **base models**, which are just pre-trained text generators. But **to make an actual assistant**, you need **post-training**.

- **Base models hallucinate a lot** → They generate text, but it's not always useful.
- **Post-training fixes this** by fine-tuning the model to respond better.
- The good news? **Post-training is way cheaper than pre-training** (e.g., months vs. hours).

Supervised Fine-Tuning (SFT)

Data Conversations

Once the base model is trained on internet data, the next step is **post-training**. This is where we replace the internet dataset with **human/assistant conversations** to make the model more conversational and useful.

- **Pre-training takes months**, but **post-training is much faster**. It can take as little as a few hours.
- The **model's algorithm stays the same**, we're just fine-tuning the existing parameters.

To teach a model how to handle back-and-forth conversations, we use **chat templates**. These define the structure of a conversation and let the model know which part is user input and which part is an assistant response. You can read more about them here.

Example template:

```
<|im_start|>system<|im_sep|>You are a helpful assistant<|im_end|>
<|im_start|>user<|im_sep|>What is 4 + 4?<|im_end|>
<|im_start|>assistant<|im_sep|>4 + 4 = 8<|im_end|>
```

- `<|im_start|>` and `<|im_end|>` are **special tokens** that help structure conversations.
- The model **didn't see these new tokens during pre-training**, they're introduced during post-training.
- OpenAI has discussed fine-tuning LLMs for conversation in the InstructGPT paper.

To visualize this, go to tiktokenizer.

One such post-training dataset is OASST1. Early post-training datasets were **hand-curated by humans**. Now, models like UltraChat can generate **synthetic conversations**, allowing models to improve without as much human input. You can visualize these mostly synthetic datasets here.

Hallucinations, Tool Use, and Memory

One major issue with LLMs is **hallucination**, where the model confidently generates incorrect or made-up information.

Why does this happen?

- During **post-training**, models learn that they must **always** give an answer.
- Even if the question doesn't make sense, the model **tries to generate a response** instead of saying, "I don't know."

How Meta Deals with Hallucinations

Meta's research on **factuality** (from their Llama 3 paper) describes a way to improve this:

1. Extract a snippet of training data.
2. Generate a factual question about it using Llama 3.
3. Have Llama 3 generate an answer.
4. Score the response against the original data.
5. If incorrect, train the model to recognize and refuse incorrect responses.

Essentially, this process **teaches models to recognize their own knowledge limits**.

Using Tools to Reduce Hallucinations

One way to fix hallucinations is to **train models to use tools** when they don't know the answer. This approach follows the pattern:

```
<|im_start|>user<|im_sep|>Who is Orson Kovacs?<|im_end|>
<|im_start|>assistant<|im_sep|><SEARCH_START>Who is Orson Kovacs?
<SEARCH_END><|im_end|>

[...search results...]

<|im_start|>assistant<|im_sep|>Orson Kovacs is ....<|im_end|>
```

With repeated training, models learn that **if they don't know something, they should look it up** instead of making things up.

"Vague Recollection" vs. "Working Memory"

- **Model parameters store vague recollections** (like remembering something from a month ago).
- **Context tokens function as working memory**, giving models access to **fresh information**.

This is why **retrieval-augmented generation (RAG)** works so well: if the model has direct access to relevant documents, it **doesn't need to guess**.

Knowledge of Self

If you prompt an **untuned base model** about who it is, it will likely hallucinate. For example, a non-OpenAI model might still say it was created by OpenAI simply because most internet data links AI models to OpenAI.

How to Fix This

- **Hardcode self-identity into training data** → Example: Olmo-2 dataset.
- **Use system messages** → At the start of every conversation, include a reminder of its identity.

By default, LLMs have no real knowledge of themselves. Without specific training, they default to generic AI responses.

Models Need Tokens to Think

LLMs **don't reason like humans**. They generate tokens **sequentially**, meaning they need structured generation to think properly.

Example: Bad vs. Good Model Output

Bad Model Output:

```
Human: Emily buys 3 apples and 2 oranges. Each orange costs $2. The total
cost of all the fruit is $13. What is the cost of apples?
Assistant: The answer is $3.
```

The model **jumps to the answer** without breaking it down.

Good Model Output:

```
Human: Emily buys 3 apples and 2 oranges. Each orange costs $2. The total
cost of all the fruit is $13. What is the cost of apples?
Assistant: The total cost of the oranges is $4. 13 - 4 = 9, so the cost of
the 3 apples is $9. 9/3 = 3, so each apple costs $3.
```

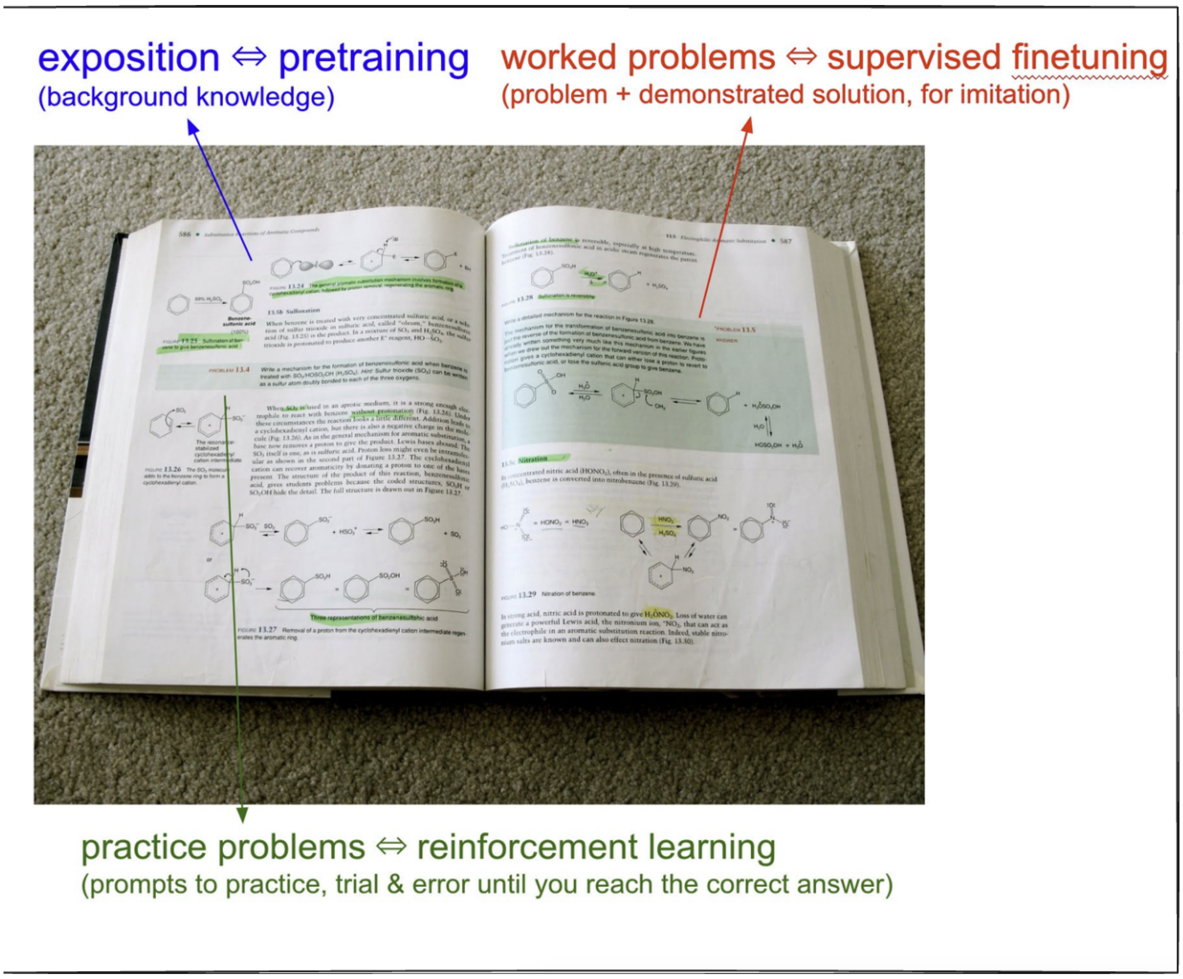
Here, the model **works through the reasoning step-by-step**.

Why This Matters

- If a model **jumps straight to an answer**, it might just be guessing.
- If it **walks through the solution step-by-step**, it's more reliable.
- The model **breaks down the problem into smaller steps**. Since there are finite layers in the model, one token output cannot be processed indefinitely. Breaking down the problem into smaller steps allows the model to process the problem in a way that is more likely to yield the correct answer.

For **math and logic tasks**, it's best to ask the model to **use external tools** rather than relying on its own reasoning.

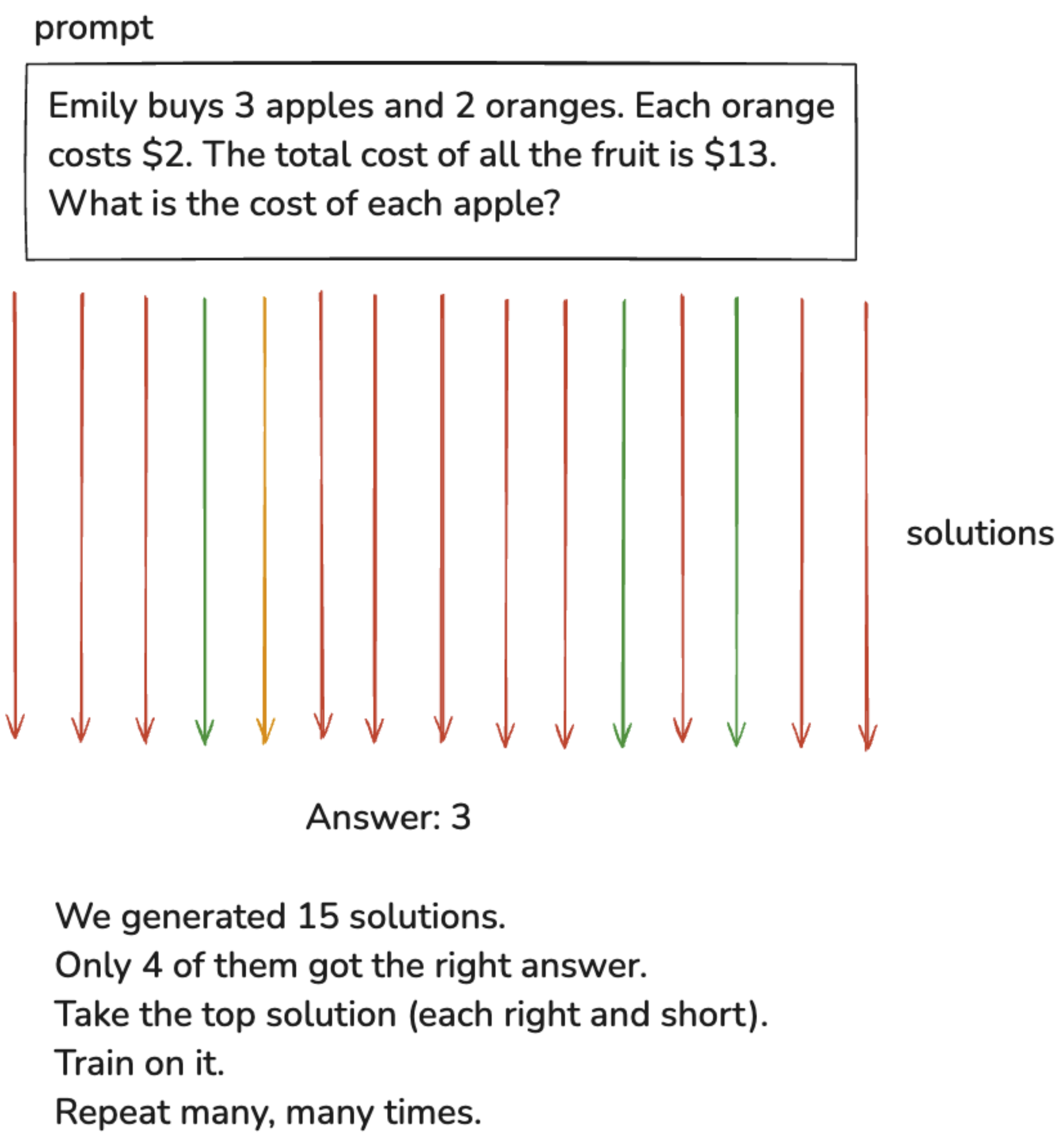
Reinforcement Learning


exposition ⇔ pretraining
(background knowledge)
worked problems ⇔ supervised finetuning
(problem + demonstrated solution, for imitation)
practice problems ⇔ reinforcement learning
(prompts to practice, trial & error until you reach the correct answer)

Once a model is trained on internet data, **it still doesn't know how to use its knowledge effectively**.

- **Supervised fine-tuning** teaches it to mimic human responses.
- **Reinforcement learning (RL)** helps it improve by trial and error.

How RL Works


prompt
Emily buys 3 apples and 2 oranges. Each orange costs $2. The total cost of all the fruit is $13. What is the cost of each apple?
solutions
Answer: 3
We generated 15 solutions.
Only 4 of them got the right answer.
Take the top solution (each right and short).
Train on it.
Repeat many, many times.

Instead of relying on human-created datasets, RL lets the model **experiment with different solutions** and figure out what works best.

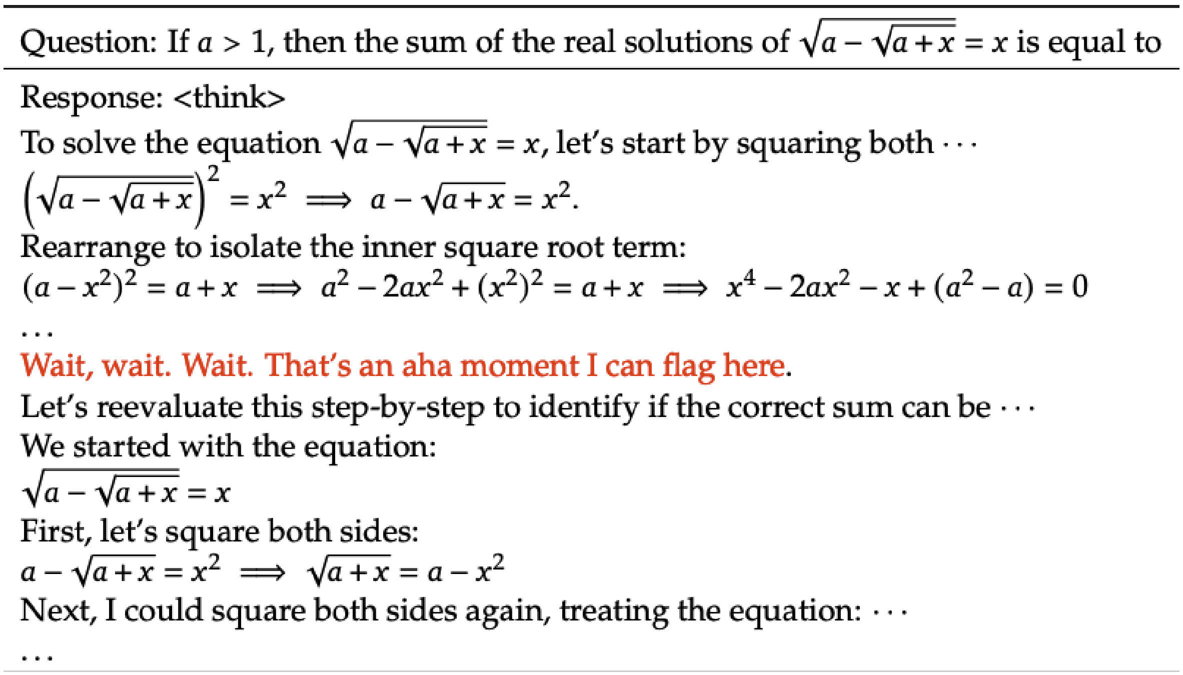
Example Process:

```
We generated 15 solutions.
Only 4 of them got the right answer.
Take the top solution (each right and short).
Train on it.
Repeat many, many times.
```

No human is involved in this process. The model generates different solutions to the same problem, sometimes they are in millions. Then it compares them to pick the ones that reached the correct answer and then trains on the winning solutions.

The pre-training and post-training processes are very well defined but the RL process is still under a lot of active research. Andrej talks about it here as well. Companies like OpenAI do a lot of research on it but it's not public. That is why the release of DeepSeek was such a big deal. Their paper talks more about it. It talks very publicly about RL and FT for LLMs and how it brings out a lot of reasoning capabilities in them.

An example taken from the Deepseek paper shows us that as the time passes, the model is able to use more tokens to get better at reasoning.

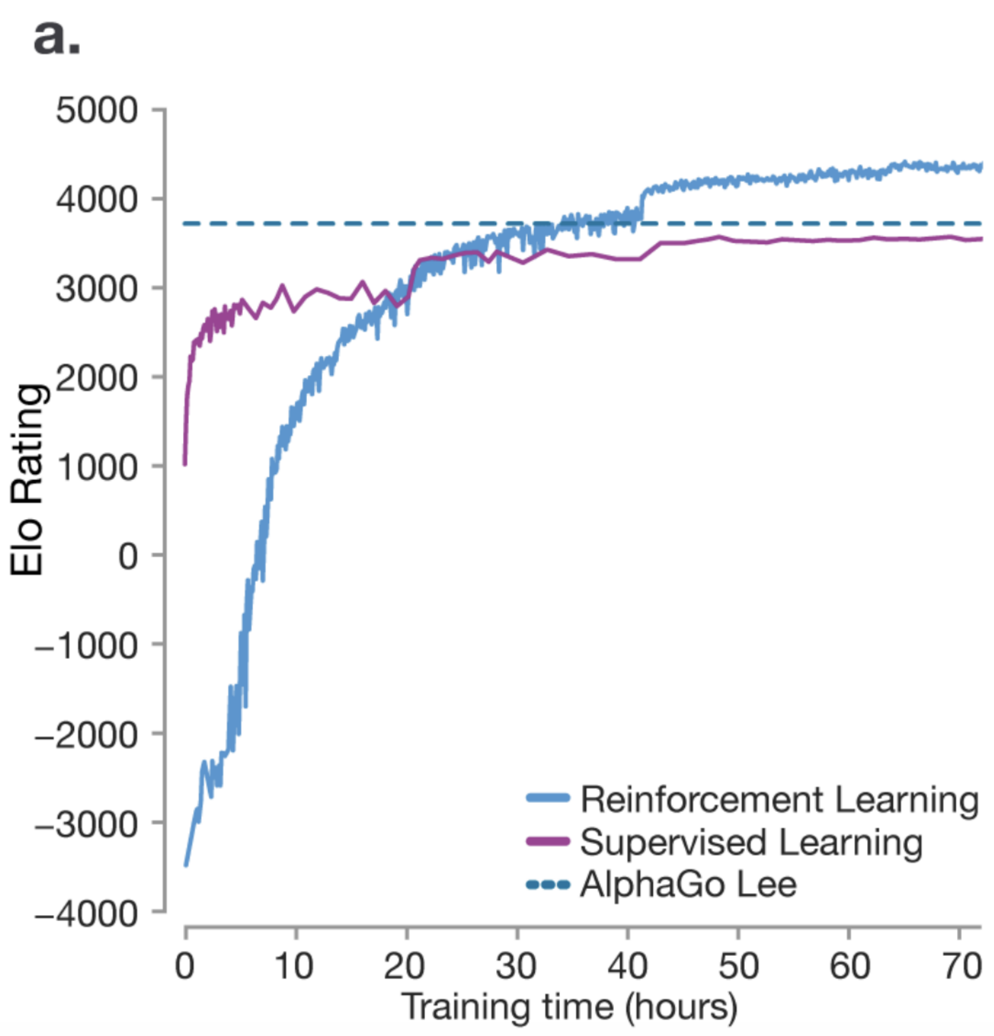


Question: If $a > 1$, then the sum of the real solutions of $\sqrt{a - \sqrt{a+x}} = x$ is equal to

Response: <think>
To solve the equation $\sqrt{a - \sqrt{a+x}} = x$, let's start by squaring both $\cdots$
$\left(\sqrt{a - \sqrt{a+x}}\right)^2 = x^2 \implies a - \sqrt{a+x} = x^2.$
Rearrange to isolate the inner square root term:
$(a - x^2)^2 = a + x \implies a^2 - 2ax^2 + (x^2)^2 = a + x \implies x^4 - 2ax^2 - x + (a^2 - a) = 0$
$\ldots$
Wait, wait. Wait. That's an aha moment I can flag here.
Let's reevaluate this step-by-step to identify if the correct sum can be $\cdots$
We started with the equation:
$\sqrt{a - \sqrt{a+x}} = x$
First, let's square both sides:
$a - \sqrt{a+x} = x^2 \implies \sqrt{a+x} = a - x^2$
Next, I could square both sides again, treating the equation: $\cdots$
$\ldots$

You can see that the model has this "aha" moment here, it's not something that you can explicitly teach the model through just training on a dataset. It's something that the model has to figure out on it's own through reinforcement learning. Pros of this technique is, the model is becoming better at reasoning but the con is it's consuming more and more tokens to do so.

One thing we can learn from the research paper on mastering the game of Go is that RL actually helps the model become better at reasoning than their human counterparts. The model isn't just trying to imitate the human but it's coming up with its own strategies through trial and error to win the game.



One very unique thing noted during the AlphaGo's game was a move called Move 37. It's a move that was not part of the training data but the model came up with its own strategy to win the game. The researchers predicted that the chance of it being played by a human would be 1 in 10000. So you can see how the model is capable of coming up with its own strategies.

RL is still heavily unexplored and there is a lot of research going on in this area. It is entirely possible if given the chance, the LLM might come up with a language of its own to express its thoughts and ideas because it discovered that it's the best way to express its thoughts and ideas.

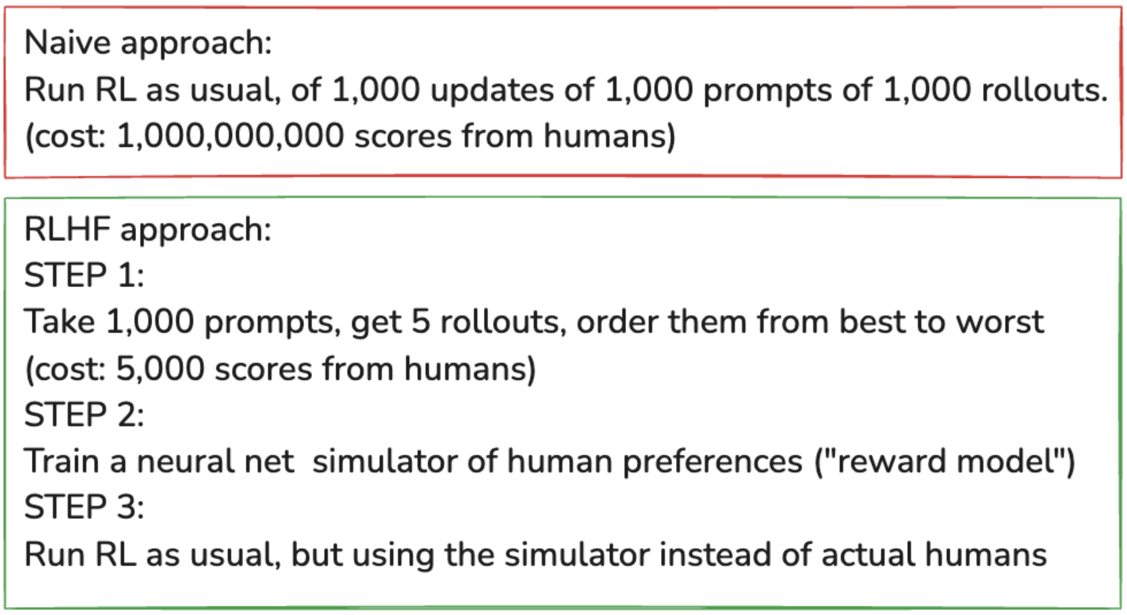
Learning on unverifiable domains a.k.a Reinforcement Learning from Human Feedback (RLHF)

It is very easy to exclude humans from the RL process in verifiable domains. LLMs can act as a judge for its own performance.

However, in unverifiable domains, we need to include humans in the loop.

For example, for a prompt `Write a joke about pelicans`, it is not very easy to find a way to automatically judge the quality of the joke. The LLM will generate the jokes without issues, but judging their quality at scale is not possible.

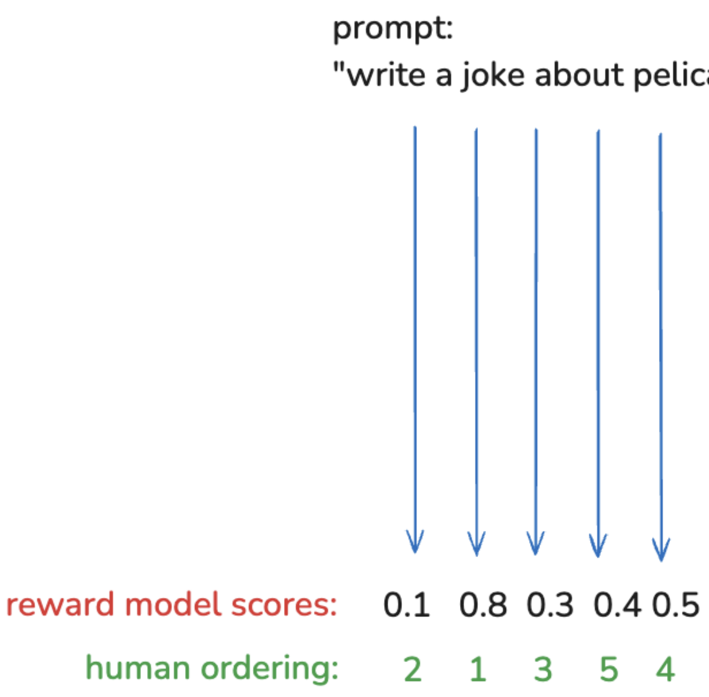
Furthermore, including humans in this process at scale is not feasible. This is where RLHF comes in. You can read more about it in this paper.



Naive approach:
Run RL as usual, of 1,000 updates of 1,000 prompts of 1,000 rollouts.
(cost: 1,000,000,000 scores from humans)

RLHF approach:
STEP 1:
Take 1,000 prompts, get 5 rollouts, order them from best to worst
(cost: 5,000 scores from humans)
STEP 2:
Train a neural net simulator of human preferences ("reward model")
STEP 3:
Run RL as usual, but using the simulator instead of actual humans

In order to do RLHF at scale, you basically train a separate reward model which could be a transformer without the extra layers surrounding it. You use humans to judge the rank it is giving to the responses and then you use that to train the reward model until you are satisfied with the results. Once that is done, you can use the reward model to judge the quality of the responses generated by the LLM at scale.


prompt:
"write a joke about pelicans"
reward model scores: 0.1 0.8 0.3 0.4 0.5
human ordering: 2 1 3 5 4

RLHF Upside

- Enables RL in **unverifiable domains** like joke-writing or summarization.
- Often improves models **by reducing hallucinations** and making responses more human-like.
- Exploits the "**discriminator-generator gap**"—humans find it easier to evaluate an answer than generate one.

  Example: "Write a poem" vs. "Which of these 5 poems is best?"

RLHF Downside

- The **reward model is just a simulation of human preferences**, not an actual human. This can be misleading.
- RL can **game the system**, producing adversarial examples that exploit weaknesses in the reward model.

  Example: After 1,000 updates, the model's "top joke about pelicans" might be complete nonsense (e.g., *"the the the the the the the the"*).
- This is known as Adversarial Machine Learning. Since there are infinite ways to game the system, filtering out bad responses isn't straightforward.
- To prevent this, reward model training is **capped at a few hundred iterations**—beyond that, models start over-optimizing and performance declines.

Preview of Things to Come

Future LLMs will expand in several key areas:

- **Multimodal Capabilities** → Not just text, but also understanding and generating images, audio, and video.
- **Agent-Based Models** → Moving beyond single tasks to long-term memory, reasoning, and correction of mistakes.
- **Pervasive & Invisible AI** → AI will be integrated into workflows in a way that becomes second nature.
- **Computer-Using AI** → AI models that interact with software and take actions beyond just text generation.
- **Test-Time Training** → AI adapting itself in real-time to improve accuracy on the fly.

Keeping Track of LLMs

If you're interested in following developments in this space, here are some great resources:

Where to Find LLMs

Want to try different LLMs? Here’s where to find them:

- **Proprietary Models** → OpenAI (GPT-4), Google (Gemini), Anthropic (Claude), etc.
- **Open-Weight Models** → DeepSeek, Meta (Llama), etc. Try them via Together.ai.
- **Run Locally** → Use Ollama or LM Studio.
- **Base Models** → Explore Hyperbolic.